APPLIED GEOPHYSICS, Vol.16, No.1 (March 2019), P. 56—63, 12 Figures.
DOI:10.1007/s11770-018-0743-8

An efficient scheme for multi-GPU TTI reverse time migration*

Liu Guo-Feng♦[1], Meng Xiao-Hong[1], Yu Zhen-Jiang[1], and Liu Ding-Jin[2]

Abstract: Reverse time migration (RTM) is an indispensable but computationally intensive seismic exploration technique. Graphics processing units (GPUs) by NVIDIA® offer the option for parallel computations and speed improvements in such high-density processes. With increasing seismic imaging space, the problems associated with multi-GPU techniques need to be addressed. We propose an efficient scheme for multi-GPU programming based on the features of the compute-unified device Architecture (CUDA) using GPU hardware, including concurrent kernel execution, CUDA streams, and peer-to-peer (P2P) communication between the different GPUs. In addition, by adjusting the computing time for imaging during RTM, the data communication times between GPUs become negligible. This means that the overall computation efficiency improves linearly, as the number of GPUs increases. We introduce the multi-GPU scheme by using the acoustic wave propagation and then describe the implementation of RTM in tilted transversely isotropic (TTI) media. Next, we compare the multi-GPU and the unified memory schemes. The results suggest that the proposed multi-GPU scheme is superior and, with increasing number of GPUs, the computational efficiency improves linearly.
Keywords: multi-GPU, kernel, peer-to-peer, forward modeling, TTI, RTM

Introduction

Seismic imaging by reverse time migration (RTM) is indispensable to seismic exploration but computationally intensive. Consequently, parallel computing studies use a variety of computing devices to accelerate the process. Graphics processing units (GPUs) with multicore architecture and high memory bandwidth delivers extremely high computing performance at reduced power and cost compared to conventional central processing units (CPUs). GPUs developed by NVIDIA® are widely used in seismic exploration because of their high performance and accessibility, e.g., prestack time migration (Liu et al.,2009,2016;Li et al.,2009;Shi et al.,2011), wave propagation forward modeling (Micikevicius,2009; Komatitsch, 2010; Okamoto et al., 2010; Weiss, 2013), RTM (Foltinek et al., 2009; Liu et al.,2009; Liu et al.,2012; Liu et al.,2013,2016; Shi et al.,2016), and waveform inversion

Manuscript received by the Editor December 8, 2017; revised manuscript received March 28, 2018.
*This work was supported by the National Key R&D Program of China(2017YFC0602204-01) and NSFC (Grant Nos. 41530321 and 41104083).
1. China University of Geosciences-Beijing, Beijing100083, China.
2. Sinopec Geophysical Research Institute,Nanjing211103, China.
♦ Corresponding author: Liu Guo-Feng (E-mail: liugf@cugb.edu.cn)
© 2018 The Editorial Department of **APPLIED GEOPHYSICS**. All rights reserved.

(Liu et al., 2012; Monton et al., 2008; Liu et al., 2015). In all these cases, the computational efficiency improved. When the datasets cannot fit into the memory of a single GPU or when increasing throughput and efficiency are required, multiple GPUs are used to process multiple tasks concurrently. Forward modeling of wave propagation and RTM operations require multiple GPUs when the computing space is larger than the memory of a single GPU. Thus, several approaches have been proposed (e.g., Nakata et al.,2011; Weiss,2013), where the data communication between different GPUs uses a percentage of the total time needed to achieve overlap of computation and communication. This means that the ideal 1:N ratio of computation time to the number of GPUs N is not reached because of the limited ability of various GPUs.

We present a series of techniques for finite-difference RTM to achieve the ideal speedup ratio using the GPU architecture. We start with forward modeling of the acoustic wave propagation equation and then apply these techniques to tilted transversely isotropic (TTI) RTM using multiple GPUs. These examples suggest that the proposed methods can be generalized for inhomogeneous operations.

Multi-GPU computation

With the development of the compute-unified device architecture (CUDA) and GPU hardware, new features have been proposed to improve the efficiency of GPU computations and to simplify multi-GPU programming. We rely on the following critical features (Cheng et al.,2014; NVidia,2017).

Concurrent kernel: "The Fermi architecture was the first complete GPU computing architecture to deliver the features required for the most demanding high-performance computation application. Fermi has been widely adopted for accelerating production workloads" (Cheng et al., 2014). One key feature of the Fermi architecture is the support for concurrent kernel execution. This means that "multiple kernels launched from the same application context are executed on the same GPU at the same time. Concurrent kernel execution allows programs that execute a number of small kernels to fully utilize the GPU" (Cheng et al., 2014). This is also the main technique used for the full overlap of communication and computation during multiple GPU implementations. The difference between serial and concurrent kernels is shown in Figure1.

Stream: "A CUDA stream refers to a sequence of asynchronous CUDA operations that are executed on a device in the order issued by the host code. A stream encapsulates these operations, maintains their ordering, and permits operations to be queued in the stream for execution after all preceding operations. These operations can include host-device data transfer, kernel launches, and most other commands that are issued by the host but handled by the device" (Cheng et al.,2014). In many cases, such as those featuring forwarding modeling and RTM, more time is spent executing the kernel than transferring data. In these situations, data communication between multiple GPUs can be completely hidden by dispatching the kernel execution and data transfer into separate streams (Figure 1).

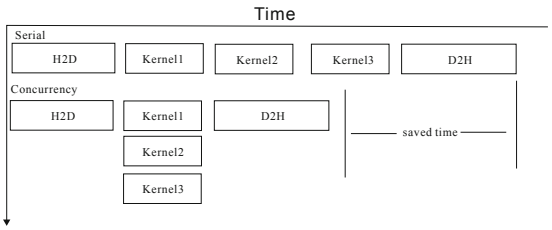

Fig.1 Serial kernel execution and concurrent kernel execution.

Peer-to-Peer (P2P) communication: When using the CUDA 4.0 or higher, kernel execution in a "device with computational capability 2.0 and higher can directly access the global memory of any GPU connected to the same PCIe root node" (Cheng et al.,2014). P2P communication supports loading and storing addresses with a CUDA kernel and across GPUs; moreover, it allows direct data copying between GPUs and data transfer is performed along the shortest PCIe path without the need to be routed through the host memory.

Forward modeling of the acoustic wave equation with multi-GPUs

We begin with the acoustic wave equation

$$\frac{\partial^2 u}{\partial t^2} = v^2\left(\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} + \frac{\partial^2 u}{\partial z^2}\right), \qquad (1)$$

where u is the pressure and v is the wave velocity in the subsurface material. The forward modeling kernel uses second-order and higher order finite-difference algorithms to approximate the time and space derivatives.

Micikevicius (2009) proved that the finite-differe-

nce computations in two dimensions leads to higher single-GPU performance. Therefore, a tiling method is proposed that uses shared memory to improve bandwidth. This is because the global memory access of GPUs is not implicitly cached by the hardware. This method can be easily extended to three dimensions, where the finite-difference calculation for each slice in the z-direction is performed using the 2D method (Liu et al.,2013).

For 3D large-scale datasets, the global memory of a single GPU cannot accommodate the calculated data. In this case, CUDA offers two strategies to deal with this condition. The first is called unified memory and it creates a pool of managed memory, where each allocated memory is accessible by multiple GPUs linked with PCIe. The second strategy uses a real multiple GPU scheme that divides the calculated space into separate GPU memory pools. These two strategies are based on the computational domain decomposition along the depth direction. The decomposition is calculated by loop statements in a single GPU. The domain decomposition is shown in Figure 2. Specifically, Figure 2a shows the computational domain that is divided into two parts with a halo region related to the finite-difference order in the depth direction. Figure 2b shows the finite difference in the x–y slice of each z, which is calculated using the Micikevicius (2009) method, and the slice in z is shown in Figure 2c.

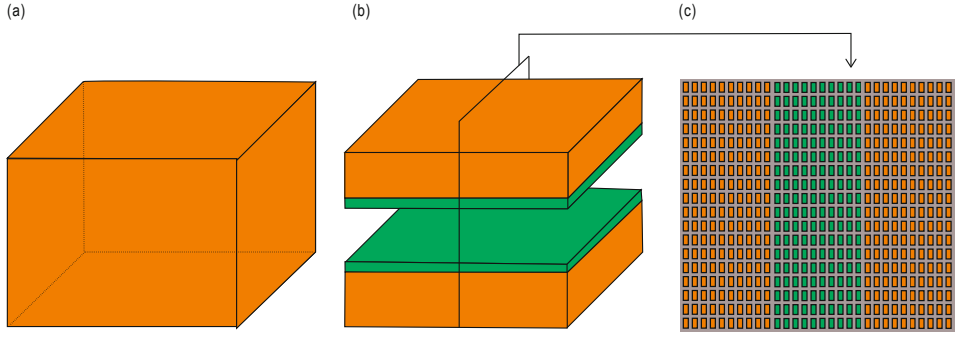

Fig. 2 Computational domain decomposition in the depth direction in multi-GPU programming: (a) computational domain, (b) domain divided into two parts with a halo region, and (c) slice along the z-direction.

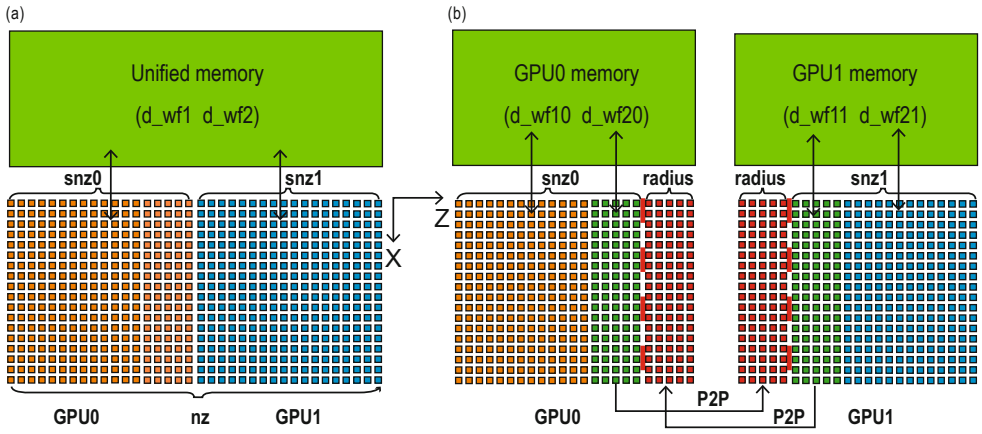

Fig. 3 Memory management for a large computational domain: (a) unified memory and (b) multi-GPU scheme.

The main difference between unified-memory and multi-GPU programming is in the memory management. In the unified-memory strategy, the computation on different GPUs operates on the same piece of memory but at different index locations. Two different GPUs use the same pointer variable at different index locations (d_wf1 and d_wf2). GPU0 is used to calculate the 3D wavefield from index iz0 = 0 to snz0, which corresponds to the orange regions in Figure 3a. GPU1 computes the remaining wavefields from index iz = snz0 + 1 to NZ, which correspond to the blue regions in Figure 3a. When a kernel is launched, the system automatically calls the corresponding data and communication between the two GPUs is automatically achieved through the

system rather than manually. This simplifies multi-GPU programming. There is no data exchange between the two devices but because every memory transaction to the mapped memory must pass over the PCIe bus, significant latency is added compared to global memory.

The multiple GPU scheme is shown in Figure 3b. The total model space is divided into two subdomains in the z-direction corresponding to GPU0 and GPU1. We concentrate on the data communication in the halo region at each time step. We also divide the space on each GPU into three parts. We consider GPU0 and calculate d_wf11 and d_wf20 separately. After finishing the finite-difference computation, d_wf20 is transferred to GPU1 using a P2P path to cover the halo region. GPU1 and GPU0 follow the same process. The size of d_wf20 and d_wf11 in the z-direction depends on the finite-difference order in the z-space domain. d_wf20 and d_wf11 are much smaller than d_wf10 and d_wf21; therefore, we create two streams for each GPU to overlap the computation and communication between the two GPUs. We then consider GPU0 to calculate the finite-difference kernel for d_wf10 in one stream, whereas the second stream first calculates the finite-difference kernel for d_wf20 concurrently with d_wf10 and then transfers the calculated d_wf20 to cover the red region in GPU1. This process, including kernel concurrency, steam, and P2P communication, (Figure 4) achieves total overlapping of computation and communication in the multi-GPU calculations. Figure 5 shows a slice along the z-direction for a shot pulse.

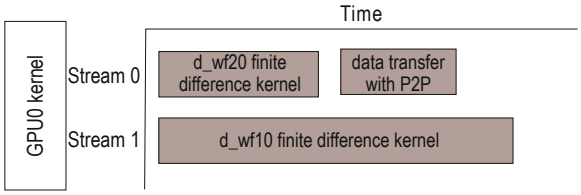

Fig. 4 Single-GPU stream in multi-GPU programming.

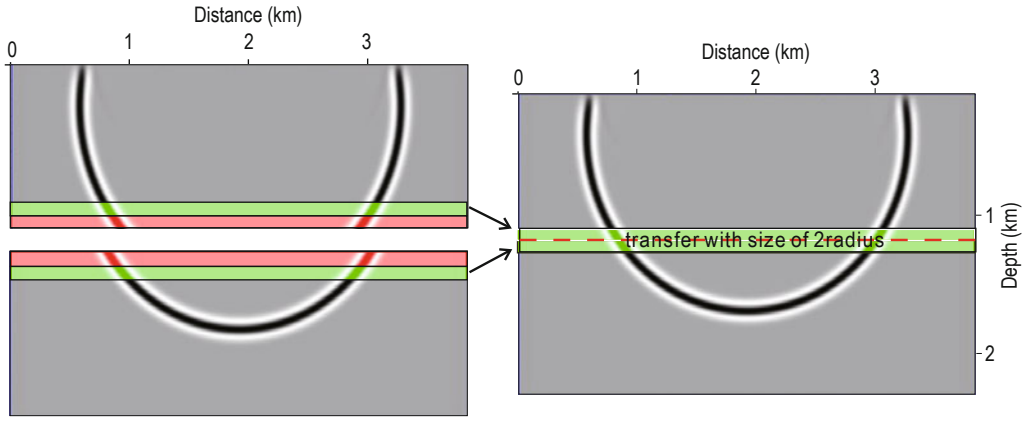

Fig. 5 Shot-pulse response slice in a homogeneous medium in multi-GPU programming.

We compare the total run time of the 3D forward modeling with different cube model volumes ranging from 304×304×304 to 1072×1072×1072 for 650 time steps using a single GPU or multi-GPUs. The workstation used had two 3.07 MHz Intel (R) Core (TM) i7 CPUs with 24 GB of DDR3 memory, a K10 with dual GK104 architecture, and 4 GB of global memory. The comparison is shown in Figure 6. The codes for the two different multi-GPU strategies show opposite acceleration effects compared to the single-GPU code. The real multi-GPU scheme, when combined with asynchronous streams, performs well and reaches the ideal speedup ratio with total overlap of communication and calculation. However, the unified memory scheme shows much lower efficiency. Figure 7 shows the speedup ratio compared with the single GPU.

TTI RTM with multi-GPUs

RTM is a powerful seismic imaging method for interpreting steep dips and subsalt regions. However, its implementation is computationally expensive. Efficiency improvements in the RTM calculations with a single GPU and mature computation schemes have been proposed, e.g., a random boundary is used to substitute for storing the wavefields in two-time wavefield extrapolation as well as sharing the GPU memory to enlarge the data access bandwidth (Liu et al., 2012; Liu et al., 2013). When the imaging space is too large to be allocated in a single-GPU memory, multiple GPUs can be used. Thus, we focus on how to improve the

execution efficiency of multiple GPUs based on wave propagation forward modeling.

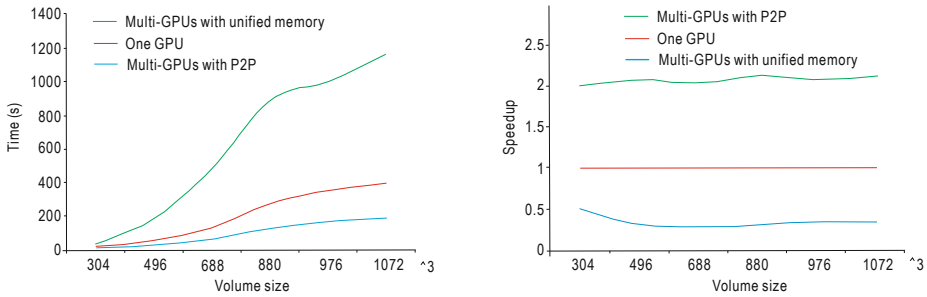

Fig. 7 Speedup ratio for the two strategies based on two GPUs with different computational volumes compared to a single GPU.

TTI RTM is a complex but more accurate method that can be used when anisotropy exists in numerous rocks and materials. We use the wave equation proposed by Fletcher (2009)

$$\begin{cases} \dfrac{\partial^2 p}{\partial t^2} = (1+2\varepsilon)V_{pz}^2 H_2 p + \alpha V_{pz}^2 H_1 q + V_{sz}^2 H_1 (p - \alpha q) & (2a) \\ \dfrac{\partial^2 q}{\partial t^2} = \dfrac{1}{\alpha}(1+2\delta)V_{pz}^2 H_2 p + V_{pz}^2 H_1 q + V_{sz}^2 H_2 (q - \dfrac{1}{\alpha} p) & (2b) \end{cases} \quad (2)$$

$$H_1 = \sin^2\theta\cos^2\varphi\frac{\partial^2}{\partial x^2} + \sin^2\theta\sin^2\varphi\frac{\partial^2}{\partial y^2} + \cos^2\theta\frac{\partial^2}{\partial z^2}$$
$$+\sin^2\theta\sin 2\varphi\frac{\partial^2}{\partial x\partial y} + \sin 2\theta\sin\varphi\frac{\partial^2}{\partial y\partial z} + \sin 2\theta\cos\varphi\frac{\partial^2}{\partial x\partial z},$$
$$H_2 = \frac{\partial^2}{\partial x^2} + \frac{\partial^2}{\partial y^2} + \frac{\partial^2}{\partial z^2} - H_1,$$

where p is the pressure wavefield, q is the auxiliary wavefield, ε and δ are the anisotropic Thomsen parameters, Vpz is the P-wave velocity in normal to the symmetry plane, Vsz is the SV-wave velocity normal to the symmetry plane, H_1 and H_2 are derivative operators, θ is the dip angle of the symmetry axis measured from the vertical, and φ is the azimuth of the symmetry axis. Note that the five parameters needed to implement TTI RTM are ε, δ, θ, φ, and Vpz. Vsz is controlled by parameter σ and is

$$\sigma = \frac{V_{pz}^2}{V_{sz}^2}(\varepsilon - \delta), \quad (3)$$

where α is a formal parameter. Following Fletcher (2009), 3D TTI RTM was implemented using Eq. (2) with α = 1 and σ = 0.75. Fletcher demonstrated that the SV velocity corresponding to σ = 0.75 removes triplications from the SV-wave front and minimizes the anisotropic term of the SV reflection coefficient.

The TTI RTM calculation has two parts. The first part comprises the source and received wavefield extrapolation by finite-difference methods, and the second comprises the imaging by cross-correlating these two wavefields at each extrapolated time step. The TTI RTM wavefield extrapolation is a little more complex than the acoustic situation where the main wavefield p and auxiliary wavefield q are extrapolated alternately. This is shown in Figure 8.

The multiple GPU extrapolation of the source and receiver wavefields follows a similar scheme, as the one introduced with the forward modeling multi-GPU kernel. The large imaging space is divided into two sections along the depth direction, as shown in Figure 9, one is for GPU0 and the other for GPU1. Each section also has three parts: the halo region, the streams, and P2P transaction features. These parts are used to overlap

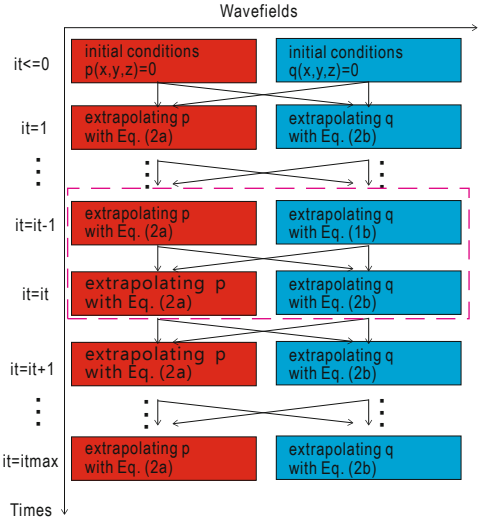

Fig. 8 Calculation framework for TTI RTM with dual wavefields p and q.

the time for computing and communicating between the two different GPUs.

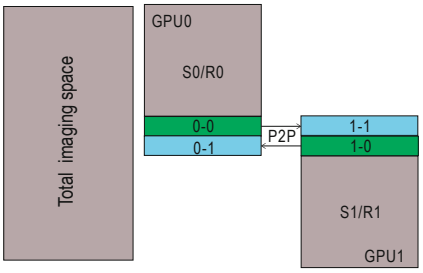

Fig. 9 Imaging space divided into two GPUs in TTI RTM wavefield extrapolation.

When cross-correlating the imaging, the wavefields are correlated at the equivalent latest time positions (it+1). For convenience, the conventional wavefields in the latest time step (it+1) are replaced by the wavefields in the current time step (it). Therefore, there is no need to wait for the extrapolation to end during each time step, and the imaging calculation is performed concurrently with the extrapolation. In TTI RTM, the extrapolation of the source and receiver wavefields is not interrelated and it can be executed concurrently.

Based on the above, we established ten streams that allow us to achieve overlapping of the various parts of the calculation. GPU0 and GPU1 each has five streams for executing the source- and receiver-wavefield extrapolations concurrently, as well as transferring data with P2P and imaging with cross-correlation. The two GPUs follow the same process at the different parts of the imaging space (Figure 10). In GPU0, for example, for the source extrapolation, Stream1 is responsible for calculating the subdomain 0-0 and transferring the result to 1-1 of GPU1 using P2P communication. Stream2 calculates the main parts of the imaging space using GPU0. At the same time, Stream3 and Stream4 are established to calculate the receiver extrapolation concurrently with the source extrapolation, and Stream5 executes the cross-correlation concurrently at time t instead of time t+1.

We applied GPU TTI RTM to a 3D TTI salt model, which is synthesized by using forward modeling. The model grid size is 901×901×250. The corresponding grid interval is 15×15×10 m^3. The number of shots in the 3D TTI salt dataset is 3134. The shot interval is 120 m. The number of channels is 16,081 and the receiver interval is 30 m. The total recording time is 8 s and the sampling rate is 4 ms. Figure 12 shows two sublines from the TTI RTM imaging results, as produced by the model. We only present the velocity and dip angle of the model parameters. The reflectors or layers are imaged well, i.e., they are continuous and strong.

Figure 11 shows that the CPU code accelerates significantly and, compared with the dual-GPU code, the efficiency increases by a factor of two. The test results verify the proposed strategy with multiple GPUs and its ability to achieve the ideal speedup ratio of 2:1.

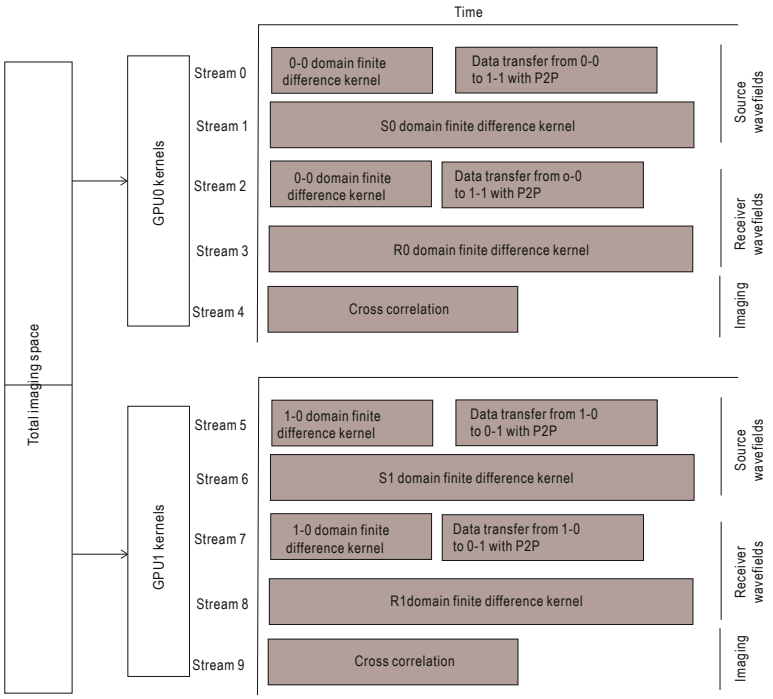

Fig. 10 Stream structure in TTI RTM with two GPUs.

When the proposed method is extended to more than two GPUs, it only involves dividing the finite-difference domain to multiple parts. However, there are no data transformations between them and the calculation time can be reduced proportionally to the number of GPUs.

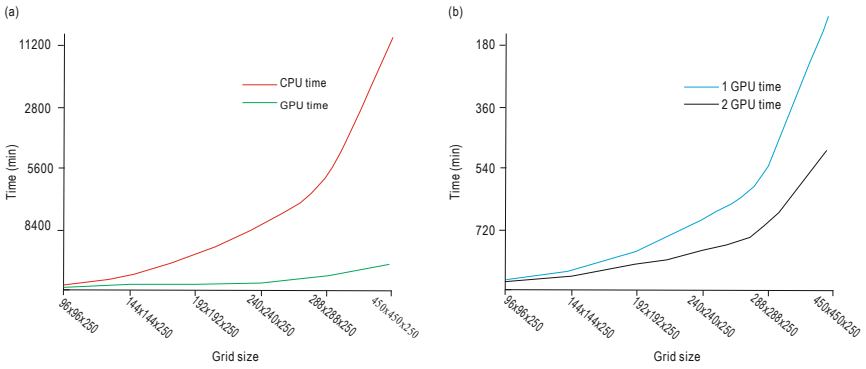

Fig. 11 Computational efficiency: (a) CPU vs GPU and (b) one GPU vs two GPU.

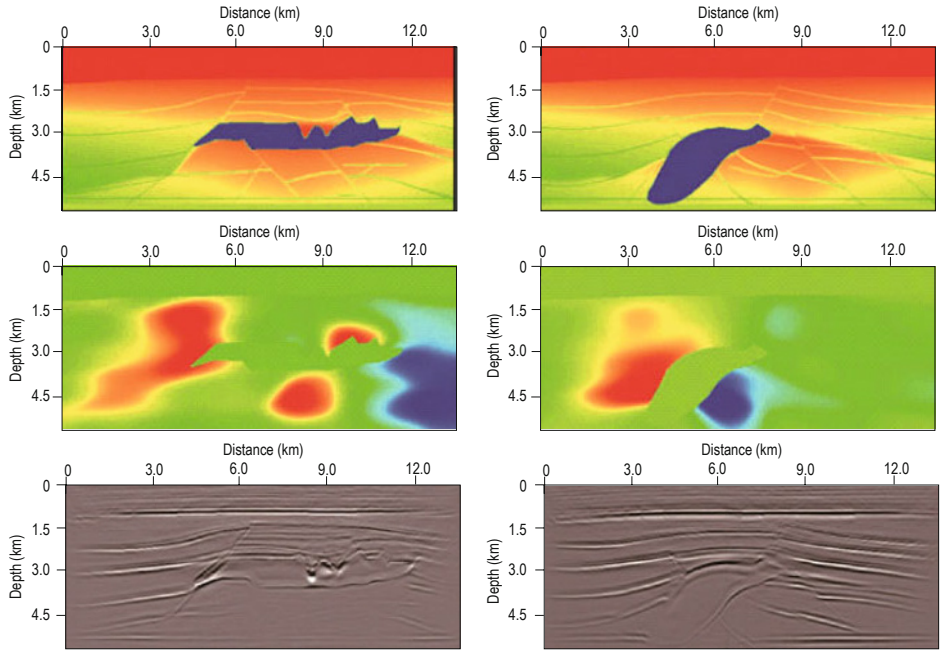

Fig. 12 The two sublines of the TTI reverse time migration results for the 3D salt TTI model. The top panels show the velocity model parameter. The middle panels show the dip angle model parameter, and the bottom panels are the imaging results based on the proposed algorithm.

Conclusions

A new multi-GPU scheme for forward modeling and RTM was proposed. Newly developed features like concurrent kernels, streams, and P2P communication were introduced. The communication time between different GPUs linked by PCIe in forward modeling is totally hidden. After changing the cross-correlation time step from (it+1) to (it), the above features were successfully applied to TTI RTM. The test results suggest that when using two GPUs for the above two modules, the computing time is reduced by half compared to one GPU. Furthermore, these techniques can easily be extended to more than two GPUs by dividing the finite-difference domain into multiple parts according to the number of GPUs. With GPU hardware development, e.g., stream processing in the GPU core and improving the bandwidth between different GPUs, the concurrent kernel and P2P communication offer the promise of efficiency improvements.

Acknowledgments

We would like to thank the reviewers, Drs. He Bingshou, Wang Boli, Li Zhenchun, and Chen Tiansheng and editors. Their comments and suggestions improved the manuscript.

References

Cheng, J., Grossman, M., and McKercher, T., 2014, Professional CUDA C Programming, Wiley & Sons Inc., Indianapolis, Indiana, P71, 73, 74, 268, 391

Foltinek, D., Eaton, D., Mahovsky, J., Moghaddam, P., and McGarry, R., 2009,Industrial-scale reverse time migration on GPU hardware: 79th Annual International Meeting, SEG, Expanded Abstracts, 2789–2793.

Fletcher R.P., X. Du, P.J. Fowler, 2009, Reverse time migration in tilted transversely isotropic media: Geophysics, 74,179-187.

Liu, H.W., Li, B., Liu, H.,Tong, X.L., Liu, Q., Wang, X. W., and Liu, W.Q., 2012. The issue of prestack reverse time migration and solutions with Graphic Processing Unit implementation, Geophysical Prospecting, **60**, 906–918, doi: 10.1111/j.1365-2478.2011.01032.x

Liu, H.W., Li, B., Liu, H., Tong, X.L. and Liu, Q., 2010. The algorithm of high order finite difference pre-stack reverse time migration andGPU implementation. The Chinese Journal of Geophysics, **53**(7), 1725–1733.

Li, B., Liu, G.F. and Liu, H., 2009. A method of using GPU to accelerate seismic pre-stack time migration: The Chinese Journal of Geophysics, 52(1), 245–252.

Liu, G.F., and Li, C., 2016, Practical implementation of prestack Kirchhoff time migration on a general purpose graphic processing unit: Acta Geophysica,**64**,1051–1063, doi: 10.1515/cgeo-2016-0033

Liu,L,R., Ding, W., Liu, H.W., and Liu, H., 2015, 3D hybrid-domain full waveform inversion on GPU: Computers and Geosciences, **83**, 27–36.

Liu, G.F.,Meng, X.H., and Liu, H., 2012, Accelerating finite difference wavefield-continuation depth migration by GPU: Applied Geophysics, **9**, 41–48

Liu,G.F.,Liu, Y.N., and Meng, X.H., 2013, 3D seismic reverse time migration on GPGPU: Computers and Geosciences, **59**, 17–23

Micikevicius, P., 2009, 3D Finite difference computation on GPUs using CUDA: Proceedings of 2nd Workshop on General Purpose Processing on Graphics Processing Units, GPGPU-2, Association for Computing Machinery, 79–84.

Morton, S., Cullison, T., and Micikevicius, P., 2008, Experiences with seismic imaging on GPUs: 70th Annual International Conference and exhibition, EAGE, Extended Abstracts, W08.

Nakata, N., Tsuji, T. and Matsuoka, T., 2011, Acceleration of computation speed for elastic wave simulation using a graphics processing unit: Exploration Geophysics, **42**, 98–104, doi: 10.1071/EG10039.

NVIDIA®, 2017, CUDA C Programming Guide; http://docs.nvidia.com/cuda/pdf/CUDA_C_Programming_Guide.pdf

Komatitsch, D., Erlebacher, G.,Göddeke, D., and Micheá, D., 2010, High-order finite-element seismic wave propagation modeling with MPIon a large GPU cluster: Journal of Computational Physics, **229**, 7692–7714, doi: 10.1016/j.jcp.2010.06.024

Okamoto, T., Takenaka, H.,Nakamura, T., and Aoki, T., 2010, Accelerating large-scale simulation of seismic wave propagation by multi-GPUs and three-dimensional domain decomposition: Earth Planets and Space, **62**, 939–942

Shi,X.H., Li, C., and Wang, S.H., 2011, Computing prestack Kirchhoff time migration on general purpose GPU: Computers & Geosciences, **37**, 1702–1710

Shi, Y and Wang, Y.H., 2016, Reverse time migration of 3D vertical seismic profile data: Geophysics, **81**(1), S31-S38

Weiss, R.M., and Shragge, J., 2013,solving 3D anisotropic elastic wave equation on parallel GPU devices:Geophysics, **78**, 7–15, doi: 10.1190/GEO2012-0063.1

Liu Guo-Feng received a B.S. in Exploration Technology and Engineering (2004) from the China University of Geosciences (Beijing), an M.S. in Earth Exploration and Information Technology (2007) from the China University of Geosciences (Beijing), and a Ph.D. in Solid Geophysics (2010) from the Institute of Geology and Geophysics, CAS. Since 2010, he has been a faculty member in the School of Geophysics and Information Technology, China University of Geosciences (Beijing) and has been a postdoctoral research fellow at the Institute of Mineral Resource, Chinese Academy of Geological Sciences. His research interests include seismic wave simulations and imaging, as well as high-performance computing.